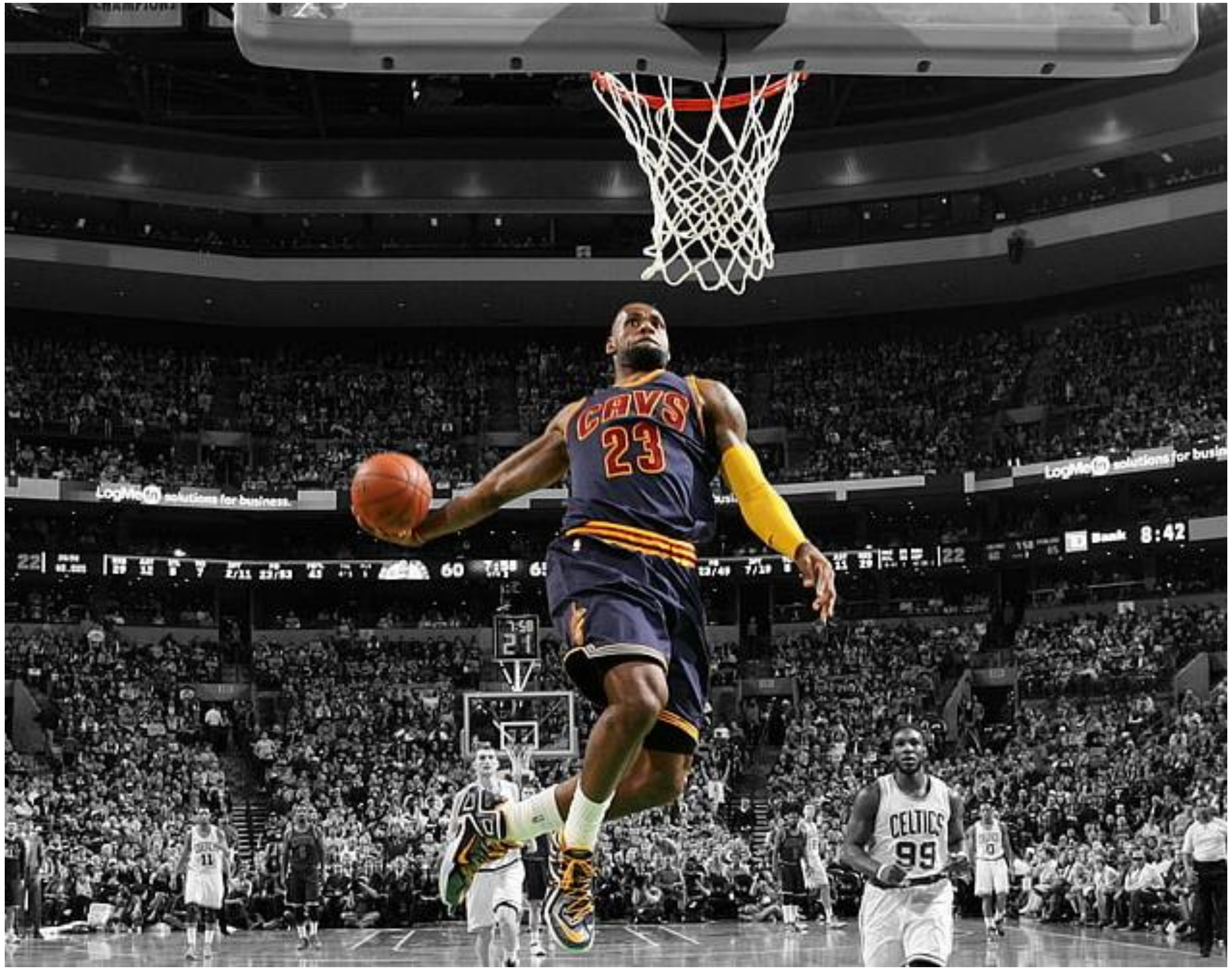
CS513 FINAL PROJECT: EXPLORATION AND MODELING OF NBA GAME OUTCOMES

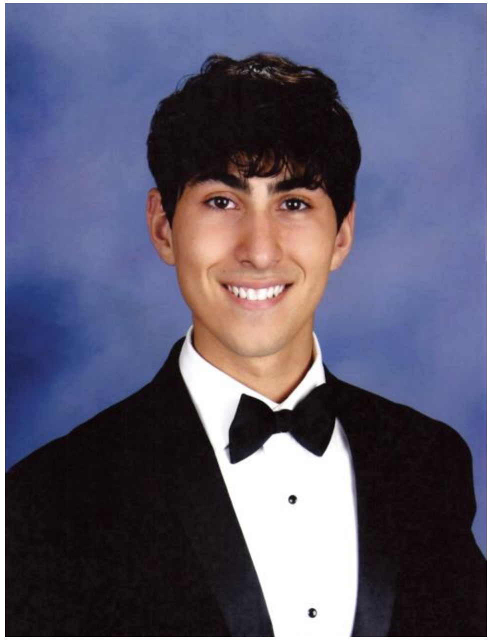
By: Sagar Patel, Nick DeRobertis,

Sarang Hadagali, Hari Patel

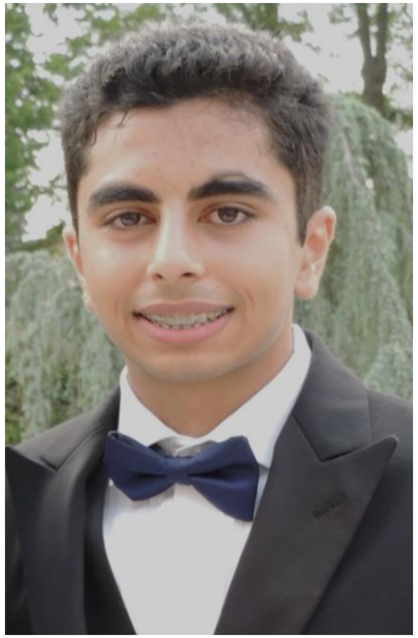


Group 21

GROUP MEMBERS INTRODUCTION



Nick DeRobertis
CWID: 20006069



Sagar Patel
CWID: 20006517

Hari Patel
CWID: 20006477

Sarang Hadagali
CWID: 20006266

PROBLEM STATEMENT

The ability to forecast professional basketball game outcomes is very useful and holds substantial significance. However, the task of predicting NBA game outcomes is very complex and is influenced by hundreds of factors ranging from important factors such as player performance and roster to external factors like refereeing and off-court distractions. We recognize this difficulty of predicting such outcomes and that is why we are trying to develop and more comprehensive and accurate approach to predict NBA game outcomes.

DATASET OVERVIEW

- Our dataset (game.csv) has over 65,000 rows and 55 columns.
  - All NBA games from the 1946-1947 to the 2022-2023 season
- We have extracted this dataset from Kaggle with a usability score of 9.41
- Our task is to use this data to predict future NBA games
- Link to dataset: https://www.kaggle.com/datasets/wyattowalsh/basketball

PRE-PROCESSING

Created a Win Streak Column to add another factor

```python
def create_streaks(nba_data):
    streak_map = defaultdict(int)
    for index, row in nba_data.iterrows():
        # Label rows
        home, away, winner = row['Home Team'], row['Away Team'], row['Winner']

        # Update team streak
        nba_data.at[index, "Home Team Streak"] = streak_map[home]
        nba_data.at[index, "Away Team Streak"] = streak_map[away]

        # Calculate streak
        streak_map[home] += 1
        streak_map[home if home != winner else away] = 0
    return nba_data
```

PRE-PROCESSING

Refigured columns to contain averages of past 5 games for each individual stat including fg pct%, rebounds, and assists

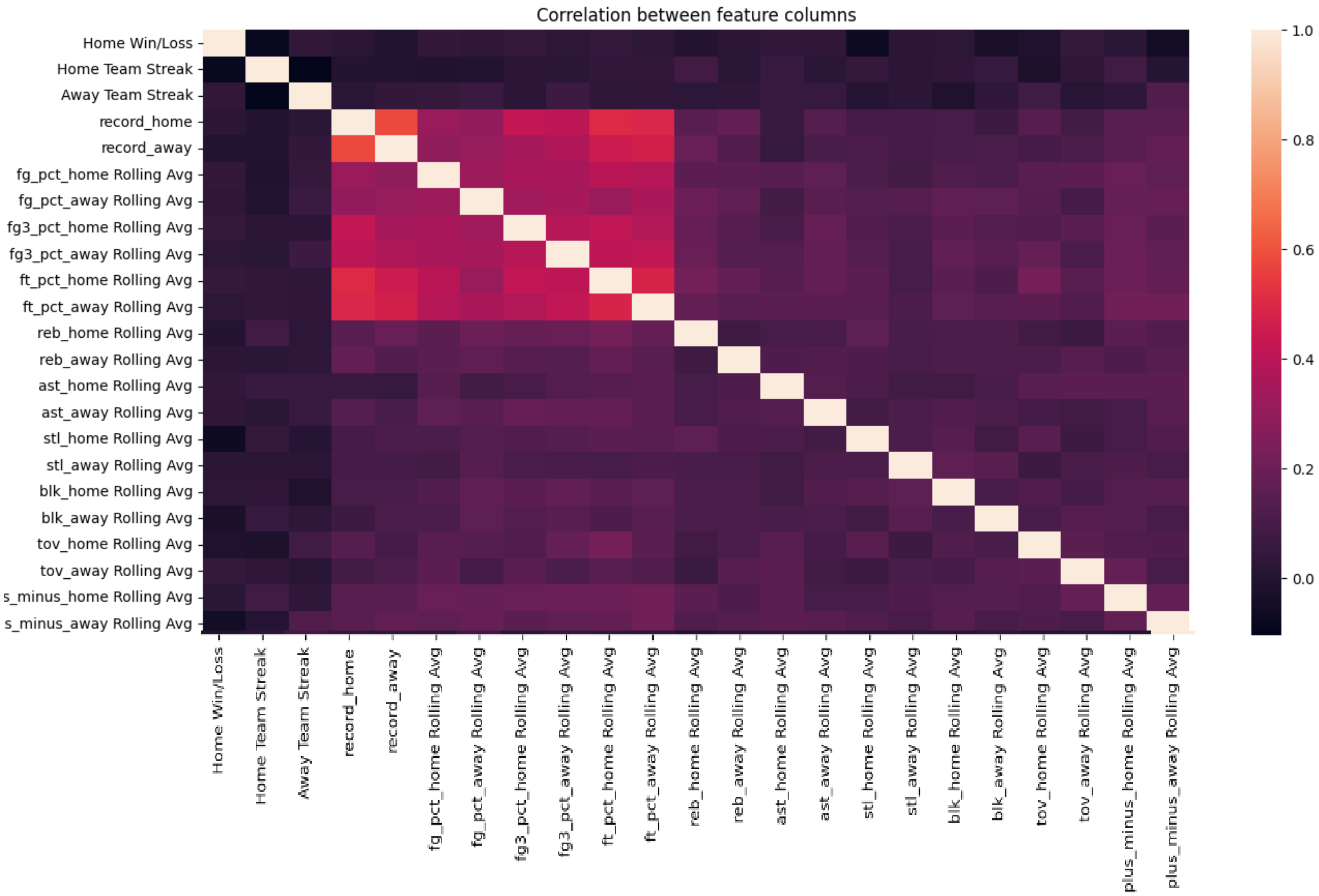
```
def rolling_avgs(nba_data, param_home, param_away):
    team_details = defaultdict(deque)
    col_home = param_home + " Rolling Avg"
    col_away = param_away + " Rolling Avg"

    for index, row in nba_data.iterrows():
        home, away = row['Home Team'], row['Away Team']
        if len(team_details[home]) == 5:
            nba_data.at[index, col_home] = sum(
                team_details[home])/len(team_details[home])
            team_details[home].popleft()
        else:
            nba_data.at[index, col_home] = row[param_home]

        if len(team_details[away]) == 5:
            nba_data.at[index, col_away] = sum(
                team_details[away])/len(team_details[away])
            team_details[away].popleft()
        else:
            nba_data.at[index, col_away] = row[param_away]

        team_details[home].append(row[param_home])
        team_details[away].append(row[param_away])
    return nba_data
```

PRE-PROCESSING

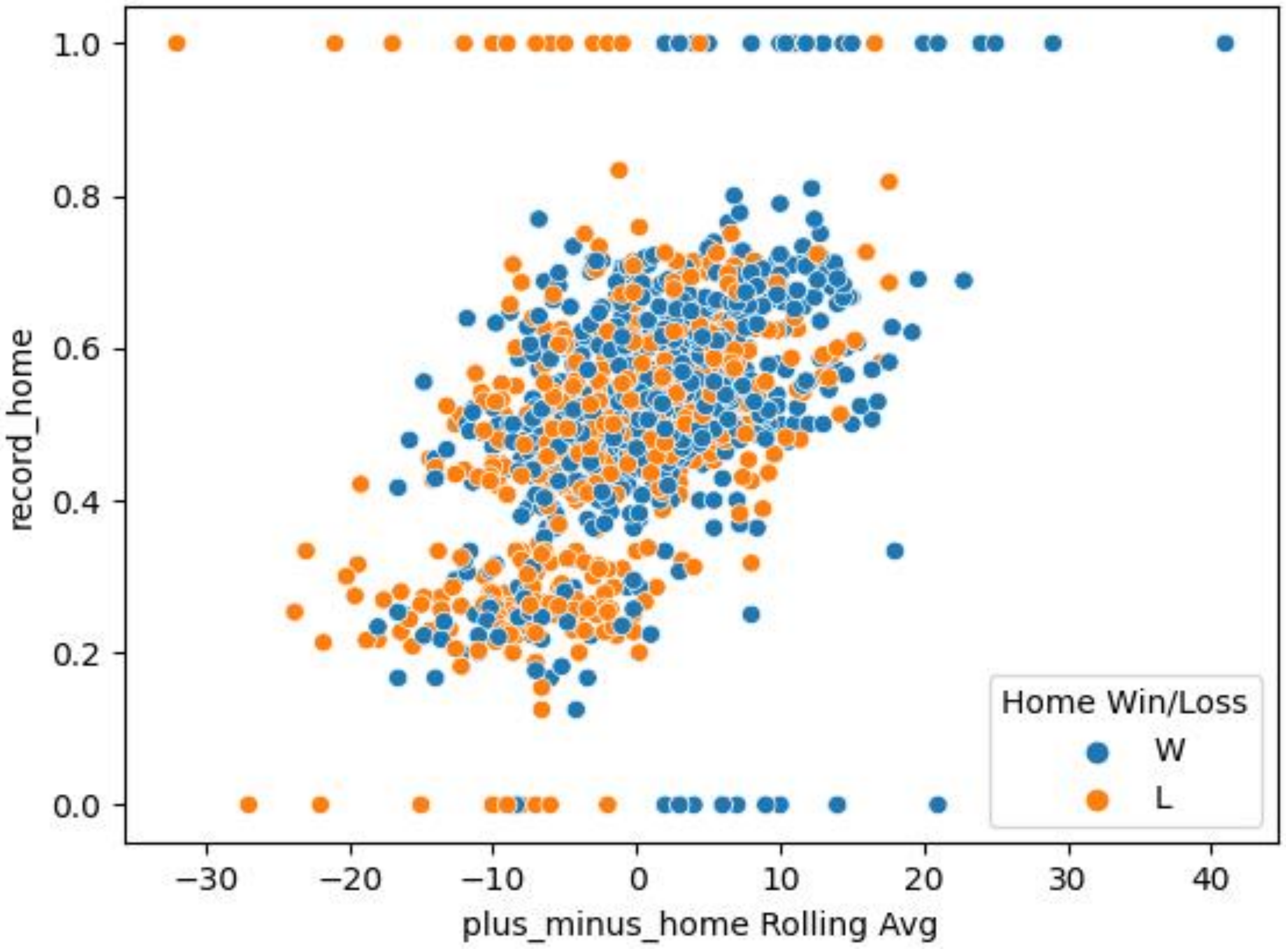
Inserted new columns which calculated the **overall records** going into a specific game

```python
def calc_records(nba_data):
    teamRecord = defaultdict(int)
    totalGames = defaultdict(int)

    for index, row in nba_data.iterrows():
        home_team = row['Home Team']
        away_team = row['Away Team']
        winner = row['Winner']

        # record_home updated
        if totalGames[home_team] == 0:
            # for first game of season, both records are 0-0
            nba_data.at[index, 'record_home'] = 0.0
        elif teamRecord[home_team] == totalGames[home_team] - 1:
            # for undefeated teams (such as 1-0 or 2-0). cant divide by zero so manually set?
            nba_data.at[index, 'record_home'] = 1.0
        else:
            nba_data.at[index, 'record_home'] = teamRecord[home_team] / \
                totalGames[home_team]

        # record_away updated
        if totalGames[away_team] == 0:
            # for first game of season, both records are 0-0
            nba_data.at[index, 'record_away'] = 0.0
        elif teamRecord[away_team] == totalGames[away_team] - 1:
            # for undefeated teams (such as 1-0 or 2-0). cant divide by zero so manually set?
            nba_data.at[index, 'record_away'] = 1.0
        else:
            nba_data.at[index, 'record_away'] = teamRecord[away_team] / \
                totalGames[away_team]

        totalGames[home_team] += 1
        totalGames[away_team] += 1

        if winner == home_team:
            teamRecord[home_team] += 1
        elif winner == away_team:
            teamRecord[away_team] += 1

    return nba_data
```

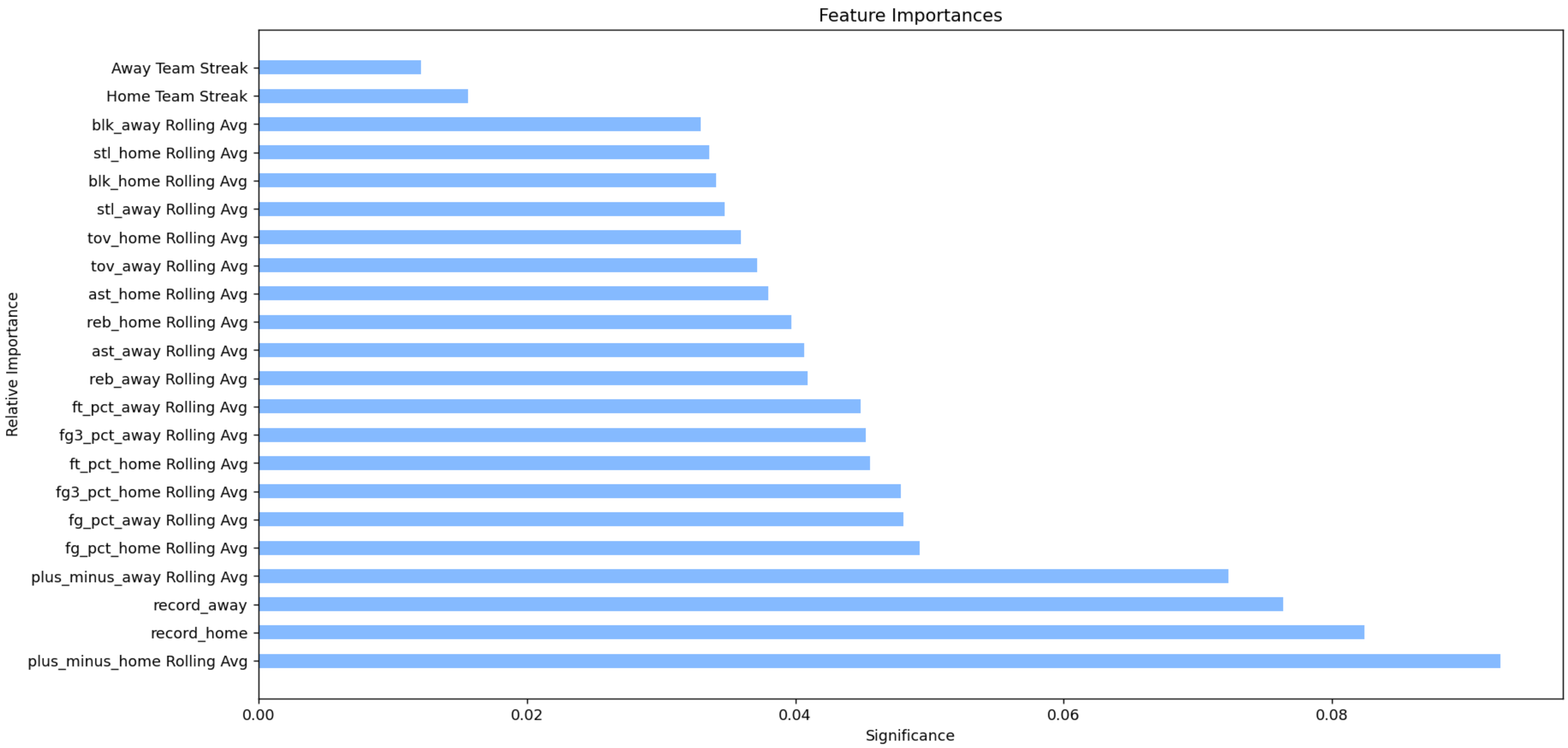
CORRELATION BETWEEN THE FEATURE COLUMNS



SCATTER PLOT OF KEY FEATURES



ANALYSIS OF FEATURE IMPORTANCES



ALGORITHMS USED

- To solve our problem, we utilized multiple classification algorithms:
  - K-nearest neighbors
  - Gaussian Naïve Bayes
  - Multinomial Naïve Bayes
  - Decision Tree Classifier (CART)
  - Random Forest
  - SVM
  - Logistic Regression

K-NEAREST NEIGHBORS

```
K Nearest Neighbors Confusion Matrix:
 [[455 606]
 [498 928]]

K Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

        Loss       0.48      0.43      0.45      1061
         Win       0.60      0.65      0.63      1426

    accuracy                           0.56      2487
   macro avg       0.54      0.54      0.54      2487
weighted avg       0.55      0.56      0.55      2487
```

GAUSSIAN NAÏVE BAYES

```
Gaussian Naive Bayes Confusion Matrix:
 [[546 515]
 [457 969]]

Gaussian Naive Bayes Classification Report:
              precision    recall  f1-score   support

        Loss       0.54      0.51      0.53      1061
         Win       0.65      0.68      0.67      1426

    accuracy                           0.61      2487
   macro avg       0.60      0.60      0.60      2487
weighted avg       0.61      0.61      0.61      2487
```

MULTINOMIAL NAÏVE BAYES

```
Multinomial Naive Bayes Confusion Matrix:
 [[   0 1061]
 [   0 1426]]

Multinomial Naive Bayes Classification Report:
              precision    recall  f1-score   support

        Loss       0.00      0.00      0.00      1061
         Win       0.57      1.00      0.73      1426

    accuracy                           0.57      2487
   macro avg       0.29      0.50      0.36      2487
weighted avg       0.33      0.57      0.42      2487
```

DECISION TREE CLASSIFIER (CART)

```
CART Confusion Matrix:
 [[511 550]
 [565 861]]

CART Classification Report:
              precision    recall  f1-score   support

        Loss       0.47      0.48      0.48      1061
         Win       0.61      0.60      0.61      1426

    accuracy                           0.55      2487
   macro avg       0.54      0.54      0.54      2487
weighted avg       0.55      0.55      0.55      2487
```

RANDOM FOREST

```
Random Forest Confusion Matrix:
 [[ 461  600]
 [ 388 1038]]

Random Forest Classification Report:
              precision    recall  f1-score   support

        Loss       0.54      0.43      0.48      1061
         Win       0.63      0.73      0.68      1426

    accuracy                           0.60      2487
   macro avg       0.59      0.58      0.58      2487
weighted avg       0.60      0.60      0.59      2487
```

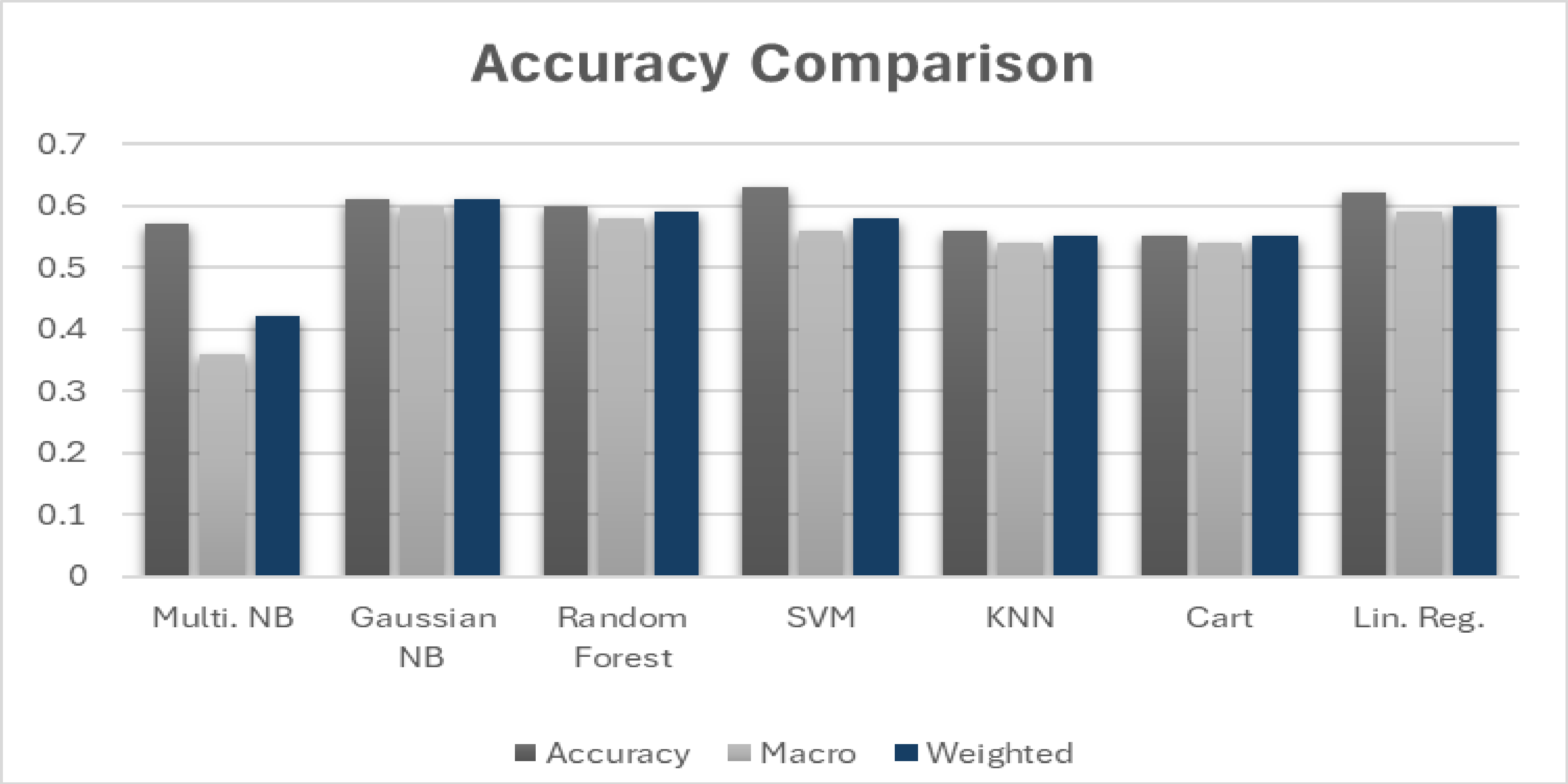
SUPPORT VECTOR MACHINE (SVM)

```
SVM Confusion Matrix:
 [[ 357  704]
 [ 256 1170]]

SVM Classification Report:
              precision    recall  f1-score   support

        Loss       0.58      0.34      0.43      1061
         Win       0.62      0.82      0.71      1426

    accuracy                           0.61      2487
   macro avg       0.60      0.58      0.57      2487
weighted avg       0.61      0.61      0.59      2487
```

LOGISTIC REGRESSION

```
Logistic Regression Confusion Matrix:
 [[ 435  626]
 [ 329 1097]]

Logistic Regression Classification Report:
              precision    recall  f1-score   support

        Loss       0.57      0.41      0.48      1061
         Win       0.64      0.77      0.70      1426

    accuracy                           0.62      2487
   macro avg       0.60      0.59      0.59      2487
weighted avg       0.61      0.62      0.60      2487
```

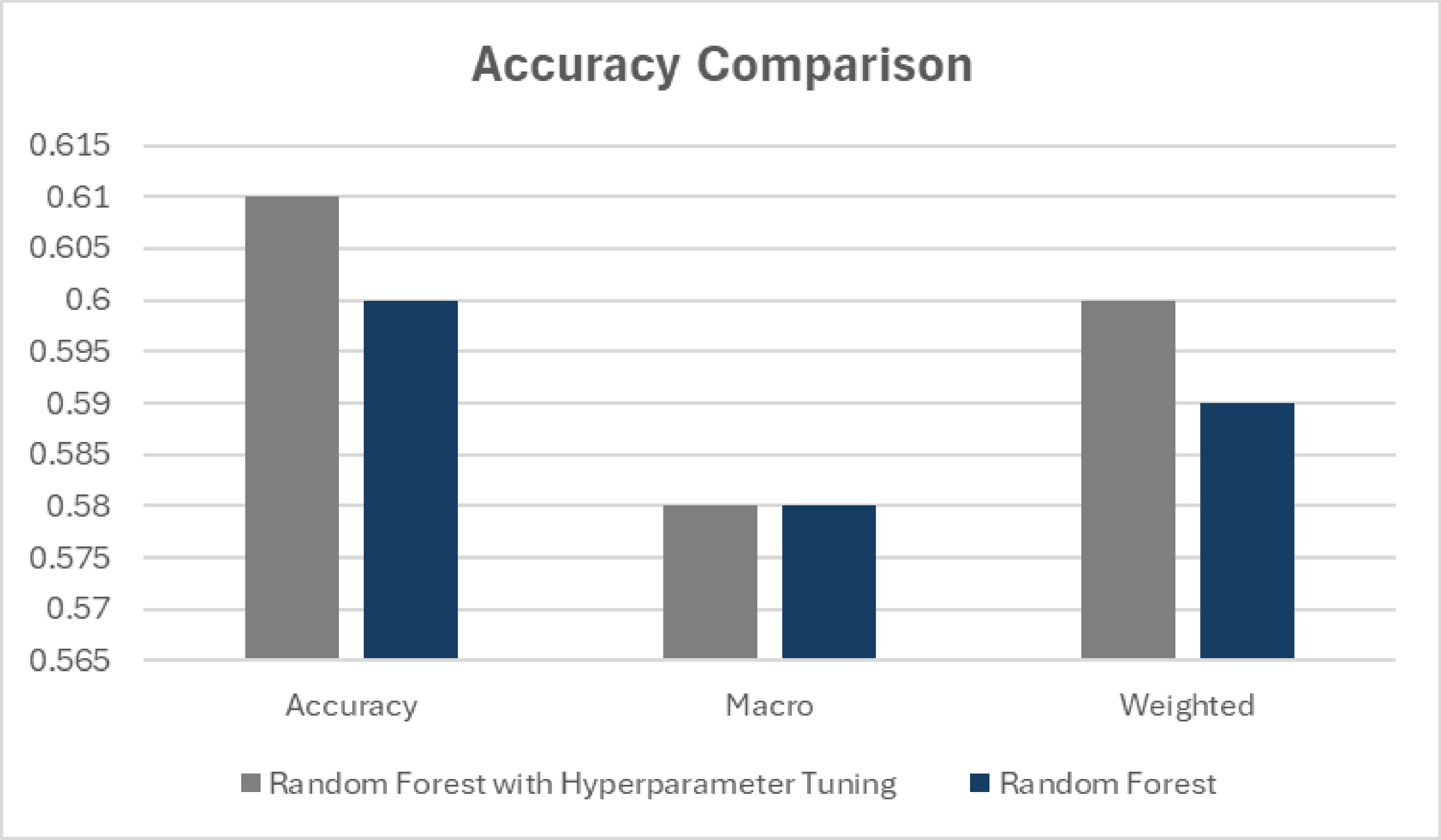
ACCURACY COMPARISON



RANDOM FOREST AFTER HYPER PARAMETER TUNING

```
Random Forest After Hyper Parameter Tuning Confusion Matrix:
 [[ 419  642]
 [ 328 1098]]

Random Forest After Hyper Parameter Tuning Classification Report:
              precision    recall  f1-score   support

        Loss       0.56      0.39      0.46      1061
         Win       0.63      0.77      0.69      1426

    accuracy                           0.61      2487
   macro avg       0.60      0.58      0.58      2487
weighted avg       0.60      0.61      0.60      2487
```

RANDOM FOREST WITH HYPERPARAMETER TUNING



CONCLUSION

- After thorough analysis and predictive modeling, the most important factors were Teams Plus/Minus, Teams Record and Teams Field Goal Percentage
  - Realistically, statistics that are correlated with wins will predict wins.
- The most accurate models were Random Forest, Gaussian Naïve Bayes and Logistic Regression with accuracies 0.61, 0.61 and 0.64 respectively
- Although, with extensive preprocessing and testing/tuning various models, the accuracies were lower than anticipated.
  - Not separating data into different seasons
  - Team players getting traded